VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Knihovna funkcí pro vykreslování grafů v jazyce Python

A Chart Programming Library in Python

2009

Radek Daníšek

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 7. května 2009 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2009 .....

Rád bych poděkoval Ing. Janu Gaurovi za čas, informace, které mi poskytnul, ochotu a pomoc při řešení zadaného problému.

Abstrakt

Vyhodnocování různých statistických dat je mnohem jednodušší, pokud jsou reprezentovány pomocí grafů. Vytvoření knihovny, která zprostředkovává vykreslování grafů na webových stránkách je obsahem této práce. V práci jako celku je obsažen i průzkum stávajících dostupných řesení. Návrh a implementace knihovny pro vykreslování kvalitně vypadajících grafů na webových stránkách je realizováno v programovacím jazyce Python.

Klíčová slova: alfa kanál, barevný přechod, graf, HSL barevný prostor, instance, knihovna, link, modul, Python, RGBA formát, tuple, ukazatel na instanci

Abstract

Evaluating of any static data is simpler, when the data is representated by graph. Creation of library, which mediate drawing graphs on a web pages is in content of this bachelor thesis. In this bachelor thesis is also investigation of current available sollutions. Project and implementation of library for drawing high-quality looking graphs on web pages are written in programming language Python.

Keywords: alpha channel, chart, gradient, HSL color space, instantion, library, link, module, Python, RGBA format, reference to instantion, tuple

Seznam použitých zkratek a symbolů

API	–	Application programming interface
PIL	–	Python Imageing Library
RGBA format	–	R (red), G (green), B (blue), A (alpha) format
HSL	–	H (hue), S (saturation), L (light) color space
PNG	–	Portable Network Graphics
VB	–	Visual basic

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Jako téma pro bakalářskou práci jsem si vybral tvorbu knihovny funkcí pro vykreslování grafů v jazyce Python. Součástí práce je i průzkum stávajících řešení, kterému je věnována 2. kapitola a vytvoření demonstračního webu, kterému je věnována kapitola 6. Samotná knihovna byla realizována pomocí souboru programových modulů. Nově vzniklý produkt je flexibilnější a uživatel může výsledné grafy mnohem více parametrizovat a tím ovlivňovat jejich vzhled a rozměry. Z časových důvodů je množina typů grafů implementovaných ve vytvořené knihovně omezená na grafy spojnicové, bodové, sloupcové, radarové a výsečové.

2 Stávající řešení

2.1 Obecné informace

V dnešní době existují již hotová, funkční řešení problému vykreslování grafů, a to nejen v jazyce **Python**, ale také v jiných jazycích, například v relativně mladém programovacím jazyce **C#** (čteno [síšárp]).

Na internetu je mnoho materiálů (freewarových i sharewarových), které obsahují informace o již hotových řešeních. Těmito materiály jsou většinou: tutoriály, obrázky, zdrojové kódy, někdy i archiv celého řešení.

2.2 Konkrétní stávající řešení (Pychart)

Protože součástí této bakalářské práce je také průzkum stávajícího řešení, hledal jsem materiál v podobě zdrojových kódů, které by byly k nahlédnutí. Našel jsem fungující řešení z roku 2005 a 2006 umístěné na internetových stránkách http://home.gna.org/pychart [2], které je zcela zdarma ke stažení a další možné editaci.

V tomto řešení jsou implementovány grafy sloupcové (jednoduché, skupinové), grafy spojnicové (křivky, čáry), dále pak grafy výsečové a radarové. Hlavní bod zájmu druhé kapitoly je tedy řešení z http://home.gna.org/pychart [2].

2.3 Výhody stávajícího řešení (Pychart)

1. API psáno přehlednou formou,
2. zdrojové texty obsahují dosti komentářů,
3. komentáře jsou výstižné a pochopitelné,
4. rychlost vykreslování grafů je příznivá,
5. jednoduchost.

2.4 Nevýhody stávajícího řešení (Pychart)

1. Počet modulů vůči množství grafů, které se dají vykreslit. Modulů je bez jednoho 60 (nebereme v úvahu moduly, které obsahují definice fontů - každý takový soubor s definicí fontu je složen ze dvou řádků, ale tyto dva řádky mají dohromady okolo 1500 znaků).
2. Jeden graf ke kompletnímu zpracování většinou potřebuje 5 a více modulů. Orientace v modulech pro programátora, který není seznámený s tímto řešením, není jasná.
3. Nevhodné použití názvů tříd či metod, které bývají pojmenovány i jedním písmenem (např.: T).

4. Chybí možnost definování průhlednosti.

5. Chybí barevné přechody.

2.5 Stávající řešení podle typů grafů (Pychart)

2.5.1 Sloupcové grafy

Jsou poměrně silnou stránkou. Pro jedna data existuje mnoho způsobů znázornění pomocí sloupcového grafu. Je definováno téměř 600 barev a tedy vysoká barevná variabilita. Grafy mohou být vykresleny horizontálně nebo vertikálně, skupinově nebo samostatně, dokonce v kombinaci s lineárním grafem nebo bez něj, dokonce mohou být znázorněny elegantním způsobem i záporné hodnoty a to tak, že nulová hodnota se nenalézá na samém dnu grafu jako nejnižší hodnota, ale je umístěna tak, že je za ní prostor pro mínusové hodnoty. Také pozadí sloupcových grafů má mnoho variací a to například: bílé, s linkami rovnoběžnými s osou x nebo šrafované s úhlem 45 stupňů a další.

2.5.2 Koláčové grafy

Koláčové grafy už tak variabilní jako sloupcové grafy nejsou. Dá se měnit barva výplně nebo barva ohraničení výsečí. U každé z výsečí lze definovat i řetězec (string) pro pojmenování výseče. Pozadí pro koláčové grafy není potřeba měnit, protože v tomto případě platí pravidlo, že v jednoduchosti je síla a bílé pozadí je správnou volbou. Koláčové grafy mají také nádech třetího rozměru. Tento nádech budí vrstva nakreslená defaultně šedou barvou ještě před samotným grafem s oblastí vykreslení mírně posunutou směrem doprava dolů se stejnými rozměry jako rozměry grafu.

2.5.3 Radarové grafy

Radarové grafy jsou ve své variabilitě téměř stejné jako koláčové grafy (bez nádechu třetího rozměru). Jen lze navíc kombinovat více grafů současně.

2.5.4 Spojnicové grafy

Spojnicové grafy jsou na tom s variabilitou lépe než grafy koláčové nebo radarové, to v každém případě. Je možno definovat barvu nebo způsob vykreslení spojnice mezi dvěma body grafu. Můžeme volit také způsob znázornění bodu grafu (kolečko, hvězdička, čtverec apod.) i jeho barvu. Mezi body lze v některých případech proložit křivku. Pozadí pro spojnicové grafy je také variabilní co se barvy a šrafování týče.

2.6 Ukázka použití API stávajícího řešení (Pychart)

Je zbytečné, dle mého názoru, vkládat ukázku zdrojového kódu nějakého modulu, který řeší bezprostřední otázky vykreslování. Bude lepší ukázat, jakým způsobem se dá s API,

```
from pychart import *
data = [(10, 20, 5, 5), (20, 65, 5, 5),
        (30, 55, 4, 4), (40, 45, 2, 2), (50, 25, 3, 3)]

ar = area.T(x_axis = axis.X(label = "X label"),
            y_axis = axis.Y(label = "Y label"))
ar.add_plot(bar_plot.T(label="foo", data = data,
                       fill_style = None,
                       error_bar = error_bar.bar3,
                       error_minus_col = 2,
                       error_plus_col = 3))
ar.draw()
```

Obrázek 1: Jak použít API pro vykreslení jednoduchého sloupcového grafu

tedy jeho konkrétní částí, pracovat. Obrázek 1 ukazuje jak vykreslit jednoduchý sloupcový graf.

Nestačí pouze vytvořit data a jednoduše je předat nějaké funkci. Je potřeba se postarat také o popisky a samotné vykreslení grafu neelegantním způsobem za pomocí vytváření ukazatele na instanci nešikovně pojmenované třídy **T** z modulu **area** s odpovídajícími parametry, poté za pomoci nově vytvořeného ukazatele na instanci zavolat metodu **add_plot** a předat jí požadované parametry, tedy instanci třídy **T** modulu **bar_plot** a té také předat správné parametry. Po tomto krkolomném postupu přichází na řadu volání metody, která se konečně postará o vykreslení. Touto metodou je metoda **draw()**, která se zavolá za pomoci výše definovaného ukazatele. Ačkoliv jsem u konce popisu toho, jak nakreslit jednoduchý sloupcový graf, zmíním se ještě o prvním řádku, kterého si jistě každý všimne. Tento řádek má za úkol nesmyslný import všech modulů.

2.7 Konkrétní stávající řešení (.netCharting)

Jako druhé stávající řešení bylo vybráno řešení **.netCharting**. Řešení je vytvořeno v programovacím jazyce **C#** a **VB**. Toto řešení je **sharewarové**. Není tedy možnost nahlédnout do zdrojových kódů týkajících se samotného kreslení grafů. Je však možnost nahlédnout do zdrojových kódů, které graf vytvoří.

2.8 Výhody a nevýhody stávajícího řešení (.netCharting)

Popis výhod a nevýhod je vzhledem k tomu, že není možnost nahlédnout do zdrojových kódu, velmi omezený.

Výhody:

- mnoho typů grafů,
- kvalitní řešení popisků,
- zabudovaná průhlednost,

- zabudované barevné přechody,
- možnost kreslení 3D grafů,
- možnost kombinování grafů.

Všechny tyto výhody můžeme spatřit u kteréhokoli typu grafu (krom možnosti kombinování - tato výhoda je dostupná pouze v některých případech). A proto není důvod rozebírat jednotlivé grafy zvlášť jako v předchozím rozboru stávajícího řešení. Jen zmíním typy grafů, které je možno použitím řešení .netCharting vykreslit:

1. sloupcové grafy,
2. trojúhelníkové grafy,
3. zabudovaná průhlednost,
4. zabudované barevné přechody,
5. možnost kreslení 3D grafů,
6. možnost kombinování grafů,
7. možnost nanášet do grafu obrázky,
8. nepřehlédnutelnou výhodou je vysoká variabilita grafů. Jeden typ grafu lze vykreslit mnoha způsoby (např. sloupcové grafy mají více než 60 variací).

Nevýhody:

1. o něco složitější postup při vytváření samotného grafu,
2. nutnost koupě.

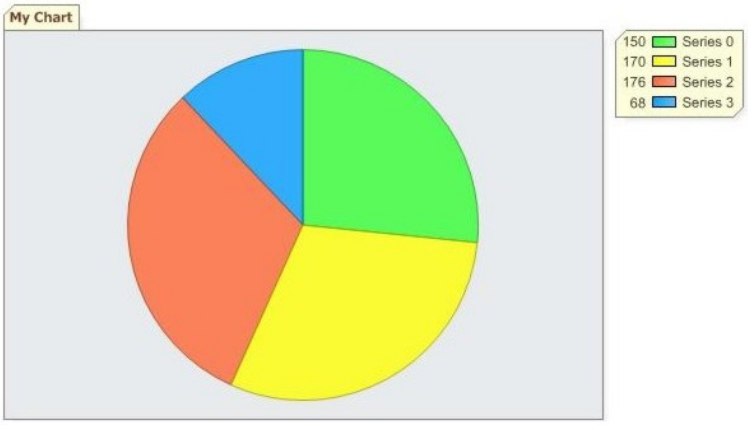
Vzhledem k tomu, že jsem neměl plnou možnost nahlédnout do zdrojových kódů, je poměr výhod / nevýhod velký a toto řešení se jeví jako velice kvalitní.

2.9 Ukázka použití API stávajícího řešení (.netCharting)

Použití API je v zásadě jednoduché a pochopitelné. Je zde však mnoho dat, které je nutno vyplnit. Ukázku použití API a její výsledek nalezneme na obrázku 2.

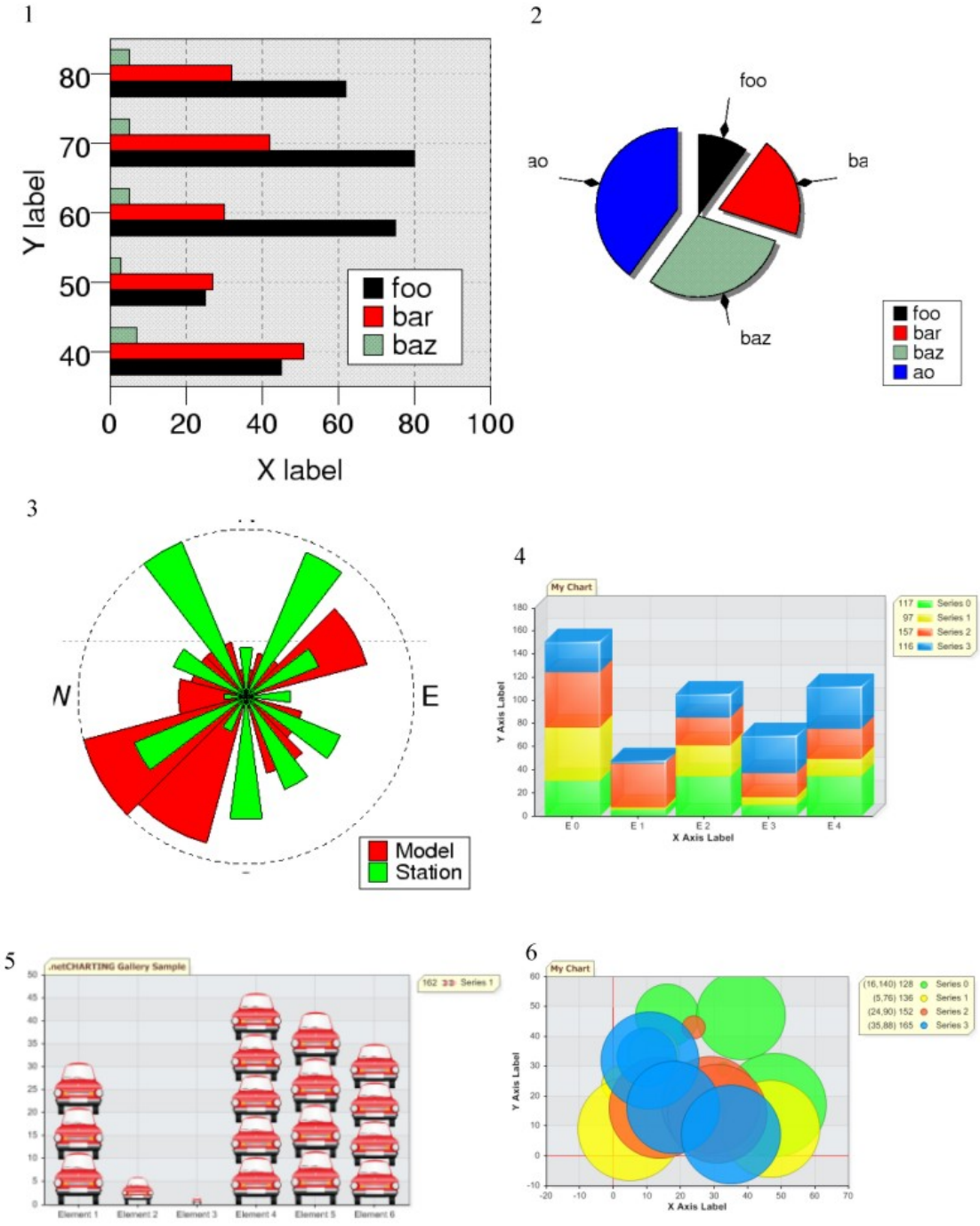
2.10 Ukázky grafů stávajících řešení (pychart a .netCharting)

Ukázky grafů 1-3 na obrázku 3 náleží řešení **pychart**, ukázky 4-6 řešení **.netCharting**.




```
<script runat="server">
SeriesCollection getRandomData(){
      SeriesCollection SC = new SeriesCollection();
      Random myR = new Random();
      for(int a = 0; a < 4; a++){
            Series s = new Series();
            s.Name = "Series " + a;
            for(int b = 0; b < 7; b++){
                  Element e = new Element();
                  e.Name = "Element " + b;
                  e.YValue = myR.Next(50);
                  s.Elements.Add(e);
            }
            SC.Add(s);
      }
      SC[0].DefaultElement.Color = Color.FromArgb(49,255,49);
      SC[1].DefaultElement.Color = Color.FromArgb(255,255,0);
      SC[2].DefaultElement.Color = Color.FromArgb(255,99,49);
      SC[3].DefaultElement.Color = Color.FromArgb(0,156,255);
      return SC;
}
void Page_Load(Object sender,EventArgs e){
      Chart.Title="My Chart";
      Chart.Use3D = false;
      Chart.Type = ChartType.Pie;
      Chart.ShadingEffect = false;
      Chart.DefaultSeries.DefaultElement.Transparency = 20;
      Chart.TempDirectory="temp";
      Chart.Width = 600;
      Chart.Height = 350;
      Chart.SeriesCollection.Add(getRandomData());
}
</script>
```

Obrázek 2: Použití API (.netCharting)



Obrázek 3: Ukázky grafů stávajících řešení

3 Teorie ke kreslení grafů

3.1 Obsah teorie

V této kapitole si objasníme:

1. princip kreslení **průhledných objektů**,
2. princip kreslení **barevných přechodů**.

3.2 Průhledné objekty

K tomu, abychom mohli kreslit průhledné objekty potřebujeme formát obrázku, který pro každý pixel umožní definovat jeho barvu a také jeho průhlednost. Vhodným formátem je **RGBA formát** (tento formát podporují např. obrázky typu **.png**). U pixelu se evidují složky **R**-červená, **G**-zelená, **B**-modrá, **A**-průhlednost.

Průhlednost se definuje pojmem **alfa kanál** (alpha channel). Základy alfa kanálu byly známy již koncem 70. let 20. století a roku 1984 byl alfa kanál zaveden do praxe. O tento zásadní krok se zasloužili **Alvy Ray Smith**, **Thomas Porter** a **Tom Duff**.

Ve 2D obrázku se uchovává barva pro každý pixel, další data jsou uložena ve složce odpovídající alfa kanálu. Hodnota těchto dat je mezi 0 a 1. Nula (0) znamená, že pixel nemá žádné informace a je úplně transparentní, tedy nemá ve své geometrii (geometrie je čtvercový objekt znázorňující pixel) žádnou barevnou plochu. Jedna (1) znamená, že je pixel neprůhledný, protože gemoetrie je plně pokryta barevnou plochou.

Díky alfa kanálu je možné zavést pojem **compositing algebra**, který definuje několik základních operací, které můžeme s obrázky provádět. Mezi tyto operace patří:

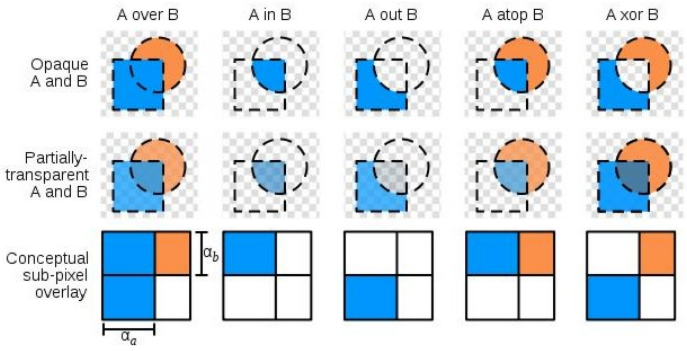
- A over B
- A in B
- A out B
- A atop B
- A xor B

Význam těchto operací a také geometrie pixelů, které touto operací vznikají je znázorněn na obrázku 4.

Pro nás nejdůležitější operací je operace **A over B**. Této operace (ve skládání obrázku) lze dosáhnout aplikací vzorce:

$$c_0 = C_a\alpha_a + C_b\alpha_b(1 - \alpha_a)$$

Kde c_0 je výsledkem operace, C_a je barva pixelu obrázku A, C_b je barva pixelu obrázku B, α_a a α_b jsou alpha složky pixelů obrázku A a B. Jestliže víme, že v obrázku je pro jednu barvu definována jedna alfa složka($c_i = \alpha_i C_i$), pak můžeme předchozí rovnici přepsat do tvaru:



Obrázek 4: Compositing algebra a její operace [5].

$$c_0 = c_a + (1 - \alpha_a)C_b,$$

kde

$$\alpha_0 = c_0/C_0 = \alpha_a + \alpha_b(1 - \alpha_a).$$

Takto vypočítaná barva však nemusí být stoprocentně odpovídající pro všechny operace, protože není asociativní. Asociativní podoba je následující:

$$c_0 = \frac{C_a\alpha_a + C_b\alpha_b(1 - \alpha_a)}{\alpha_0}$$

Tento postup je v operacích, které provádějí operace týkající se alfa kanálu, více využíván.

V samotném řešení se však hodnoty složek jednotlivých pixelů podle těchto vzorců nepočítají, je využito funkcí knihovny **PIL**. Postup pro kreslení průhledných objektů je následující:

1. Vytvoříme nový obrázek pomocí funkce **Image.new** (funkce z balíku Image) s určitou velikostí a v režimu "**L**" (režim masky, nekreslí se zde klasickou barvou, ale indikátorem průhlednosti). Do tohoto obrázku kreslíme objekty a místo barvy jim přiřazujeme osmi bytové číslo udávající jejich průhlednost.

2. Vytvoříme si cílový obrázek pomocí funkce **Image.new** (funkce z balíku Image) s určitou velikostí, barvou pozadí a režimem např. "**RGBA**".

3. Vytvoří se pomocný dočasný obrázek, do kterého kreslíme objekty, které pak budou v cílovém obrázku průhledné dle požadavků (v tomto kroce, při kreslení objektů je vhodné kreslit stejné objekty i do obrázku v režimu "**L**", tedy do masky).

4. Po dokončení kreslení následuje kompozice cílového a pomocného obrázku podle vytvořené masky průhlednosti pomocí funkce **paste(obrázek, souřadnice, maska)**. Tato funkce se volá na ukazateli na zdrojový obrázek (v našem případě **im.paste()**). Význam parametrů obrázek, souřadnice, maska je následující:

- **obrázek -** Ukazatel na obrázek (dočasný).
- **souřadnice -** X-ová a Y-ová souřadnice startovního a koncového bodu obdélníkové oblasti, do které se pomocný obrázek podle určité masky vloží do cílového obrázku. Startovní bod je levým horním vrcholem, koncový bod je pravým spodním vrcholem.
- **maska -** ukazatel na masku (obrázek ve formátu "**L**")

5. Posledním krokem je uložení cílového obrázku pomocí funkce **save** (v našem případě **im.save("cesta")**).

Z postupu je jasné, že o průhlednost se starají funkce z modulu **Image**(pro operace se samotným obrázkem - uložení, vytvoření apod.) a **ImageDraw** (kreslení různých objektů s definovanou průhledností).

Výsledkem mého postupu jsou kvalitně vypadající průhledné objekty. Postup není také náročný na čas.

3.3 Barevné přechody

3.3.1 Nejjednodušší barevné přechody

Nejprve definujeme barevný prostor, ve kterém si princip kreslení barevných přechodů vysvětlíme. Tímto prostorem je **HSL prostor**. Barevný prostor HSL je zobrazen na obrázku 5.

HSL se skládá ze tří rozměrů:

1. **Hue** - Úhel, kterým specifikujeme barvu. Tento úhel může nabývat hodnot 0 - 360 stupňů.
2. **Saturation** - Určuje saturaci (sytost) barvy. Čím má barva vyšší sytost, tím je výraznější. Saturace může nabývat hodnot 0 - 1 (0 odpovídá nula procentům, 1 odpovídá sta procentům).
3. **Light** - Určuje světlost barvy. Může nabývat hodnot 0 - 1 (0 odpovídá nejvyšší příměsi bílé bary, 1 žádné příměsi bílé barvy).

Ukázky některých barev definovaných v modelu HSL najdeme na obrázku 6.

Nejjednodušší barevné přechody lze vytvořit kreslením po sobě jdoucích barevných čar, jejichž barva se od předešlé liší vždy jen v jedné z HSL souřadnic. Rozdíl musí být dostatečně malý, aby přechod nevypadal jako několik čar vedle sebe.

- Jestliže budeme měnit první sořadnici (HSL souřadnic) každé barvy, dostaneme více odstínový barevný přechod. Nevýhodou je, že pořadí barev je předem dáno (dle barevného modelu HSl) a nelze tak vykreslit přechod kterýchkoli dvou barev.
- Budeme-li měnit druhou souřadnici (HSL souřadnic) každé barvy, dostaneme barevný přechod, který bude složen z jednoho odstínu (od méně syté barvy po sytou nebo naopak).

Obrázek 5: HSL barevný prostor [5]

RGB	HSL	HSV	Result
(1, 0, 0)	(0°, 1, 0.5)	(0°, 1, 1)	
(0.5, 1, 0.5)	(120°, 1, 0.75)	(120°, 0.5, 1)	
(0, 0, 0.5)	(240°, 1, 0.25)	(240°, 1, 0.5)	

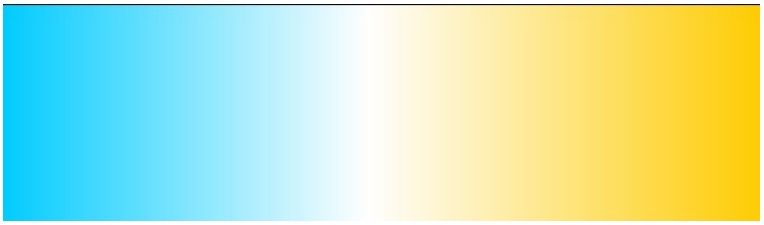
Obrázek 6: Ukázka HSL barev [5]

- Chceme-li vykreslit barevný přechod od definované barvy k bílé barvě (popř. černé), budeme měnit třetí souřadnici (zvyšovat či snižovat).

3.3.2 Složitější barevné přechody

Barevného přechodu mezi dvěma libovolnými barvami lze dosáhnout prostřednictvím třetí souřadnice. Algoritmus tvorby složitějšího barevného přechodu nejprve generuje přechod mezi první zvolenou barvou k barvě bílé (třetí souřadnice roste), pak generuje přechod od barvy bílé ke druhé zvolené barvě (třetí souřadnice se zmenšuje). Výsledkem je barevný přechod, který však obsahuje bílou barvu.

Bílou barvu lze částečně odbourat, né však úplně. Důvodem je nenávaznost dvou počátečních barev. Částečného odbourání bílé (přebytečné) barvy lze dosáhnout tak, že v algoritmu generujícím přechod od první barvy k barvě bílé urychlíme jeho kroky (rozdíly mezi barvami), ne však v celém průběhu generování přechodu, ale v určité části (např.



Obrázek 7: Složitější barevný přechod s využitím bílé barvy

od poloviny algoritmu). Efektivnějším způsobem by bylo poměrové zvyšování rozdílů kroků v závislosti na již vygenerovaném přechodu (čím větší část přechodu je nakreslená, tím větší jsou rozdíly mezi následujícími barvami).

Poměrové zvyšování rozdílů kroků však platí jen do poloviny generování barevného přechodu. Ve druhé polovině algoritmu by se stejným způsobem rozdíly mezi jednotlivými kroky zmenšovaly až do určité doby.

Možný výsledek algoritmu generujícího barevný přechod mezi dvěma barvami přes barvu bílou je zobrazen na obrázku 7.

3.3.3 Nejsložitější barevné přechody

Kvalitně vypadajícího barevného přechodu, který neobsahuje nežádoucí barvy, lze dosáhnout za pomoci využití prvku průhlednosti (tzv. **alfa kanálu**). Obecný ostup je následující:

1. Celý rozměr cílového barevného přechodu vyplníme požadovanou barvou, která má průhlednost nastavenou na nulovou hodnotu (je neprůhledná).

2. Kreslíme postupně čáry druhou barvou přechodu. Každá z těchto čar má definovaný prvek průhlednosti o jeden krok větší než čára předchozí (první čára je neprůhledná, poslední čára zcela průhledná).

V mém řešení je tento obecný postup implementován v následující podobě:

1. Připravíme první obrázek, který při vytvoření vybarvíme první barvou přechodu (pomocí funkce **Image.new(formát, velikost, barva)** z modulu **Image**).

2. Připravíme obrázek s maskou (obrázek formátu "**L**") velikosti 1x255. Takto definovaný obrázek začneme vykreslovat řádek po řádku, nejprve neprůhlednou barvou až po barvu zcela průhlednou (neprůhledná barva má hodnotu 255, průhledná má hodnotu 0).

3. Obrázek s maskou s rozměry 1x255 upravíme (pomocí funkce **im.resize** - funkce balíku **Image**) tak, aby rozměry odpovídaly rozměrům prvního obrázku.

4. Připravíme druhý obrázek, který při vytvoření vybarvíme druhou barvou přechodu.

Obrázek 8: Nejsložitější barevný přechod

5. Do prvního obrázku (pomocí funkce **im.paste** - funkce balíku **Image**) vložíme druhý obrázek podle předlohy masky.

Výsledkem takového postupu je dokonale vypadající dvoubarevný přechod neobsahující rezistentní barvy. Nejsložitější barevné přechody jsou implementovány v mém řešení. Ukázka nejsložitějšího barevného přechodu je znázorněna na obrázku 8.

4 Vlastní řešení

4.1 Prostředí vlastního řešení

Řešení je realizováno v programovacím jazyce **Python verze 2.5** za použití již hotové knihovny **Python Imageing Library** nazývané zkráceně **PIL**. Jádro programovacího jazyka včetně grafického uživatelského rozhraní a dokumentace najdeme na www.python.org [1]. Knihovnu PIL nalezneme na www.pythonware.com/products/pil [4].

4.2 Obsah vlastního řešení

Ve svém řešení jsem vytvořil **17** programových modulů, které jsou schopny vykreslovat:

- Sloupcové grafy
- Koláčové grafy
- Spojnicové grafy
- Radarové grafy
- Pyramidové grafy
- Trojúhelníkové grafy

V tabulce 1 je znázorněn přehled všech programových modulů s krátkým popisem, které se pro vykreslování grafů využívají. Všechny moduly a jejich funkčnost budou v následujících podkapitolách detailně popsány.

4.3 Požadavky na samotné řešení

Na samotné řešení bylo kladeno několik požadavků:

- Požadavek na kvalitně vykreslené grafy
- Požadavek na co nejkratší čas strávený samotným vykreslováním
- Požadavek na použití průhlednosti alespoň v některém typu grafu
- Požadavek na použití barevných přechodů v některém typu grafu
- Požadavek na objektově orientovaný způsob programování modulů

Všechny požadavky byly splněny, navíc byly přidány dva typy grafů - bodové grafy a pyramidový graf.

Jméno modulu	Stručný popis
Background.py	Modul pro vykreselní jednoduchého pozadí grafu.
Background3D.py	Modul využívající Background.py pro vykreslení prostorového pozadí grafu.
BarChart.py	Modul pro vykreslení jednoduchých sloupců.
BarChart.3D	Modul využívající BarChart.py k vykreslení jednoduchých a také prostorových grafů.
demo.py	Modul sloužící jako ukázka použití vytvořeného API, jeho obsah generuje ukázky většiny typů grafů.
GradientColor.py	Modul, který podle vstupního řetězcového (stringového) názvu barvy vrací její RGBA formát.
GradientFilePie.py	Modul, který vytváří soubor, jehož obsahem je barevný kruhový přechod a vrací ukazatel na tento soubor.
GradientFileRectangle.py	Modul, který vytváří soubor, jehož obsahem je barevný obdélníkový přechod a vrací ukazatel na tento soubor.
LineChart.py	Modul pro vykreslení spojnicových grafů.
Mask.py	Modul, který v zadaném obrázku nahradí určitou barvu za jinou.
PieChart.py	Modul pro vykreslování koláčových grafů.
PointChart.py	Modul pro vykreslování bodových grafů.
PyramidChart.py	Modul pro vykreslování pyramidových grafů.
RadarBackground.py	Modul pro vykreslování pozadí pro radarové grafy.
RadarChart.py	Modul pro vykreslování radarových grafů.
TransparentBarChart3D.py	Modul pro vykreslování průhledných sloupcových 3D grafů.
TransparentGradientBarChart3D.py	Modul pro vykreslování průhledných sloupcových 3D grafů s barevným přechodem.
TriangleChart.py	Modul pro vykreslování trojůhelníkových grafů.

Tabulka 1: Přehled modulů

4.4 Rozbor vlastního řešení dle modulů

4.4.1 Modul Background.py

Modul obsahuje třídu **BackgroundC**, která dědí ze třídy **object**. Pro vytvoření pozadí pro sloupcové, trojúhelníkové či pyramidové grafy je nutno vytvořit instanci této třídy a předat jí parametry specifikující dané pozadí. Parametry jsou: **biggestValue, pictureSize, numberValues, where, backgroundSize, depth, color1, color2, colorEdgeInside, colorEdge, pad**. Význam parametrů je následující:

- **biggestValue -** největší hodnota v grafu. Od tohoto parametru se odvíjejí prostřednictvím funkce __roundValue(x) popisky orientačních hodnot v cílovém obrázku umístěné v blízkosti levého okraje pozadí.
- **pictureSize -** velikost výstupního obrázku. Jednotky velikosti jsou pixely.
- **numberValues -** počet hodnot grafu. Pozadí je tomuto počtu optimalizováno formou vertikálního dělení na stejnoměrné části. Jednotlivé části pro jednotlivé hodnoty.
- **where -** cesta k cílovému obrázku. Možnost zadání ve formě relativní nebo absolutní vždy formou řetězce.
- **backgroundSize -** velikost pozadí ve výstupním obrázku. Jednotky velikosti jsou pixely.
- **depth -** parametr definující hloubku pozadí, tedy velikost třetího rozměru. Jednotkou je zaokrouhleně 1.4 pixelu.
- **color1 -** první barva pozadí. Tato barva je v pozadí převažující.
- **color2 -** druhá barva pozadí. Tato barva je v pozadí méně zastoupena než barva první.
- **colorEdgeInside -** barva oddělující první barvu pozadí od druhé.
- **colorEdge -** barva vyznačující levý a spodní okraj pozadí.
- **pad -** definuje odskok pozadí grafu od samotného okraje celého obrázku v pixelových jednotkách.

Pro samotné vykreslení pozadí grafu je nutno na vytvořené instanci zavolat metodu **draw()**. Třída BackgroundC obsahuje metody **draw()**, **__createFile()**, **__openFile()** a **__roundValue(x)**. Tyto metody mají za úkol (po řadě):

- Vykreslit pozadí
- Vytvořit samotný soubor, tedy obrázek, kde bude pozadí vykresleno
- Otevřít již existující obrázek

- Vypočítat zaokrouhlenou hodnotu nejvyšší hodnoty grafu

Modul také obsahuje import jiných modulů a to: **Image** a **ImageDraw**. Modul Image slouží k pohodlné práci s manipulací s obrázky, např. vytvoření obrázku, uložení apod. Modul ImageDraw je součástí PIL a slouží k pohodlnému kreslení, např. čar, obdélníků, polygonů, výsečí apod.

4.4.2 Modul BarChart3D.py & BarChart.py

BarChart3D obsahuje třídu **BarChart3DC**, která dědí ze třídy **object**. Třída BarChart3DC slouží k vykreslování sloupcových grafů, které mohou a nemusí být trojrozměrné a mohou nebo nemusí obsahovat barevné přechody, tzv. gradienty. Možné parametry pro vytvoření instance třídy BarChart3DC jsou: **values, gradientColor1, gradientColor2, where, pictureSize, backgroundDepth, color1Bg, color2Bg, colorEdgeInsidebg, colorEdgeBg, backgroundPad, barDepth**.

Pro vytvoření instance je nutno zadat pouze první z nich. Ostatní parametry se při nepoužití doplňují defaultními hodnotami. Význam parametrů je následující:

- **values -** všechny hodnoty grafu. Předává se jako tuple.
- **gradientColor1 -** první barva výplně sloupců grafu, tedy přechodu, tzv. gradientu, zadávaná jako řetězec. Pro jednobarevnou výplň sloupců je nutno zadat první a druhou barvu přechodu stejnou.
- **gradientColor2 -** druhá barva výplně sloupců grafu, tedy přechodu.
- **barDepth -** je velikost třetího rozměru sloupců. Jedna jednotka odpovídá zaokrouhleně délce 1.4 pixelu.

Ostatní parametry nebudeme vypisovat znovu, jedná se o parametry, které se použijí při tvorbě pozadí, co který znamená bylo řečeno v modulu 4.4.1.

Pro vytvoření grafu je nutné následně na instanci zavolat metodu **draw()**, která vytvoří instanci třídy **BackgroundC**, předá jí požadované parametry, zavolá na ní metodu **draw()** (třídy BackgroundC) a tím vykreslí požadované pozadí.

Pak použije všechny popsané parametry při vytváření instance třídy **BarChartC** z modulu BarChart. Na této instanci zavolá metodu **draw()** (třídy BackgroundC) a tím vykreslí přední stranu sloupců do grafu. Posledním krokem je vykreslit třetí rozměr zavoláním funkce **__drawZDimension(listing)** třídy BarChart3DC. Parametrem listing je seznam x-ových a y-ových souřadnic startovních a koncových bodů jednotlivých předních stran sloupců grafu. Tento seznam generuje a vrací jako návratovou hodnotu funkce draw() třídy BarChartC.

BarChart.py importuje moduly: **Image**, **ImageDraw** a z modulu **GradientFileRectangle** importuje třídu **GradientFileRectangleC**, která má za úkol vykreslit barevný přechod dvou specifikovaných barev (v zadané velikosti) do souboru a odkaz na výsledek vrátit jako návratovou hodnotu (použití v metodě **__drawBar(coordinates, yCoordinateOfText, value)** třídy BarChartC).

BarChart3D.py importuje: **Image**, **ImageDraw**, z modulu Background importuje třídu **BackgroundC**, z modulu BarChart importuje třídu **BarChartC** a z modulu GradientColor importuje třídu **GradientColorC**. Význam jednotlivých modulů byl popsán výše.

4.4.3 Modul GradientColor.py

Modul obsahuje třídu **GradientColorC**, která dědí ze třídy **object**. Tato třída podle vstupního řetězcového názvu barvy vrací její RGBA formát (0 - 255, 0 - 255, 0 - 255, 0 - 255). Jediným vstupním parametrem při vytváření instance této třídy je **color**, tedy řetězcový název barvy.

GradientColor.py obsahuje jednu metodu - **draw()**, která vrací RGBA formát požadované barvy.

GradientColor.py importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1) a **string**. String obsahuje funkce pro snadnou práci s řetězci.

4.4.4 Modul GradientFilePie.py

Modul obsahuje třídu **GradientFilePieC**, která dědí ze třídy **object**. Tato třída vykresluje podle parametrů specifikovaný barevný přechod, specifikované velikosti. Barevný přechod má koláčový tvar. Při vytváření instance této třídy musíme předat následující parametry: **color1, color2, size**. Význam parametrů je následující:

- **color1 -** je první barva barevného přechodu
- **color1 -** je druhá barva barevného přechodu
- **size -** je velikost výsledného obrázku s přechodem (př. (800, 600) - v pixelech).

Modul obsahuje jednu metodu - **draw()**, která vykreslí požadovaný barevný přechod.

GradientFilePie.py dále importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1) a **string** (4.4.3).

4.4.5 Modul GradientFileRectangle.py

Modul obsahuje třídu **GradientFileRectangleC**, která dědí ze třídy **object**. Tato třída vykresluje podle parametrů specifikovaný barevný přechod specifikované velikosti. Barevný přechod má tvar obdélníku. Při vytváření instance této třídy musíme předat následující parametry: **color1, color2, size**. Význam parametrů je stejný jako u vytváření instance třídy GradientFilePieC (4.4.4).

GradientFilePie.py dále importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1) a **string** (4.4.3).

4.4.6 Modul LineChart.py

Modul obsahuje třídu **SimpleLineChartC** a **OrientedLineChartC**. Nejprve popíšeme první z nich. **SimpleLineChartC** dědí ze třídy **object**. Úkolem této třídy je vykreslit podle zadaných parametrů spojnicové grafy. Při vytváření instance této třídy máme k dispozici nemálo parametrů na specifikaci požadovaného grafu. Parametry jsou následující: **values, chartColor, chartPointColor, backgroundColor, where, pictureSize, axisColor, axisWidth, chartWidth, chartPointWidth a pad**. Význam těchto parametrů je následující:

- **values -** hodnoty grafu, zadávané v pořadí, v jakém mají být vykresleny (bereme v úvahu hlavně spojnice mezi nimi), formou zápisu ((x1,y1), (x2, y2), ...).
- **chartColor -** definuje barvu spojnic mezi jednotlivými hodnotami, tedy mezi jednotlivými body.
- **chartPointColor -** definuje barvu jednotlivých bodů.
- **backgroundColor -** definuje barvu pozadí grafu. Zde je nutno dodat, že pozadí je jiného typu, než pozadí např. pro sloupcové grafy, a proto se definuje pouze jedna barva místo čtyř barev.
- **axisColor -** definuje barvu os souřadnicového systému.
- **axisWidth -** definuje šířku os souřadnicového systému.
- **chartWidth -** definuje šířku spojnice mezi jednotlivými hodnotami, tedy mezi jednotlivými body grafu.
- **chartPointWidth -** definuje šířku průměru jednotlivých bodů grafu.
- **where, pictureSize, pad -** mají stejný význam jako parametry v modulu Background.py (4.4.1).

Chceme-li vykreslit samotný graf po vytvoření instance třídy SimpleLineChartC (stačí zadat pouze parametr values - ostatní parametry jsou nepovinné) s příslušnými parametry, zavoláme na této instanci metodu **draw()**. Tato metoda vykreslí graf a těsně před skončením zavolá funkci **__eraseUsefull()**. Je to poslední funkce, kterou třída SimpleLineChartC obsahuje a plní funkci úklidu, tj. vymaže části grafu, které zasahují do definovaného odskoku pozadí od okraje obrázku a jsou tedy nežádoucí.

LineChart.py obsahuje ještě třídu **OrientedLineChartC**, která se od SimpleLineChartC liší jen druhem hodnot, se kterými skutečně pracuje. Tato třída má za úkol vykreslovat různé spojnicové grafy, které dokáží znázornit kladné i záporné hodnoty a střed os souřadnicového systému se uživateli jeví, jako by měl souřadnice (0, 0). V SimpleLineChartC neměl střed os souřadnice (0, 0), ale (šířka obrázku / 2, výška obrázku / 2). Chceme-li vykreslit spojnicový graf pomocí OrientedLineChartC, musíme po vytvoření instance zavolat metodu **draw()**. Třída však obsahuje ještě další dvě pomocné metody: **__recalculateValuesForPrint(size, values)** a **__max()**. Význam těchto dvou metod je následující:

- **__recalculateValuesForPrint -** přepočítává orientované (uživatelem definované) hodnoty na neorientované (hodnoty v rámci skutečného obrázku) hodnoty.

- **__max -** slouží k nalezení největší hodnoty, ať kladné či záporné, v uživatelem definovaných hodnotách a podle největší hodnoty vypočítá velikost dočasného obrázku, do kterého se graf zakresluje a v závěrečné fázi je pak velikost tohoto obrázku upravena na velikost požadovaného pozadí a je vložen do výstupního obrázku.

Modul LineChart.py importuje dále: **Image** (4.4.1), **ImageDraw** (4.4.1) a z modulu GradientColor třídu **GradientColorC** (4.4.3).

4.4.7 Modul Mask.py

Modul obsahuje třídu **MaskC**, která dědí ze třídy **object**. Tato třída nahrazuje pixely definované jejich barvou (v modulu - **color**) jednoho zdrojového souboru (v modulu - **imTarget**) za pixely druhého zdrojového souboru (v modulu - **imInput**), avšak se neprocházejí všechny pixely souboru a kontroluje se podmínka, zda barva pixelu odpovídá definované barvě pro nahrazení, ale je zde ještě jeden parametr (**coordinates**), který vymezuje oblast, na kterou se maska zaměřuje a tudíž se první zdrojový soubor nemusí procházet celý. Výsledný obrázek, tedy po dokončení nahrazování pixelů, je uložen zpátky na místo prvního zdroje (v modulu imTarget). Cesta, kde se obrázek uloží, se specifikuje v posledním parametru **whereToSave**, kde se očekává cesta souborového systému (může být relativní, nebo absolutní).

MaskC obsahuje pouze jednu metodu **compositeMask(imTarget, imInput, color, whereToSave, coordinates)**. Význam popsán v předešlém odstavci.

Mask.py importuje pouze jeden modul, kterým je modul **Image** (4.4.1), ze kterého jako jeden ze dvou jedinných modulů v celém řešení využívá metody pro **výběr a vložení pixelu** (druhým je modul BarChart3D (4.4.2), který využívá metodu výběru pixelu k bočnímu stínování sloupců).

4.4.8 Modul PieChart.py

Modul obsahuje jednu třídu **PieChartC**, která dědí ze třídy **object**. Tato třída má za úkol vykreslování koláčových grafů podle definovaných parametrů, které použijeme zejména při vytváření instance této třídy. Možné parametry jsou: **values, colors, pictureSize, circleSize, where, backgroundColor a pad**.

Kromě circleSize a colors zde není asi žádný parametr, který ještě nebyl popsán (4.4.1). **CircleSize** definuje průměr koláčového grafu jako celku, **colors** definuje množinu barev pro graf. Ostatní parametry mají stejnou funkčnost, jako stejnojmenné parametry, které již byly dříve popsány až na jednu výjimku, parametr **pictureSize** - až do teď byla velikost obrázku definována dvourozměrně (šířka, výška). V PieChart.py je pictureSize definována jako jeden rozměr. Důvodem je, že koláčový graf má vždy tvar kruhu, je tedy stejně široký jako vysoký, proto je i výsledný obrázek stejně široký jako vysoký a pro definování rozměrů stačí jeden rozměr.

Chceme-li vykreslit specifikovaný graf, zavoláme na vytvořené instanci třídy PieChartC (stačí zadat pouze parametr values - ostatní parametry jsou nepovinné) metodu **draw()**. Tato metoda pro vykreslování používá další pomocné metody: **__calculateAngle(fullangle, value, sum)** a **__allvalues()**. Význam je následující (po řadě):

- **__calculateAngle -** slouží k vypočítání velikosti výseče pro jednotlivé hodnoty, přesněji řečeno, vypočítá velikost úhlu, který výseč zabírá s ohledem na velikost plného úhlu (360 stupňů) ve stejném poměru, jako jedna hodnota ku sumě všech hodnot.

- **__allvalues -** slouží k sumarizaci všech hodnot.

PieChart.py dále importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1), **math** a z modulu GradientColor importuje třídu **GradientColorC** (4.4.3). Všechny moduly již byly popsány krom modulu math. Modul math implementuje mnoho užitečných matematických funkcí. Modul PieChart z modulu math používá funkce **sqrt(x)** a **sin(x)**. Význam těchto funkcí je následující:

- **sqrt(x) -** vrací ze zadaného čísla jeho druhou odmocninu.

- **sin(x) -** vrací sinus zadaného úhlu. Úhel se však nezadává ve stupních, jak by se dalo předpokládat, ale v jednotkách radiánů.

4.4.9 Modul PointChart.py

Modul obsahuje třídu **PointChartC**, která dědí ze třídy **object**. Tato třída generuje požadované bodové grafy podle definovaných parametrů, které se použijí při vytváření instance. Použitelnými parametry jsou: **values, biggestPointWidth, where, pictureSize, axisColor, axisWidth, backgroundColor a pad**. Jejich význam následující:

- **values -** stejnojmenný parametr byl mnohokrát popisován, avšak v modulu PointChart má poněkud jiné vlastnosti. V tomoto parametru se krom samotných hodnot grafu definuje pro každou hodnotu zvlášť x-ová a y-ová souřadnice výskytu hodnoty v grafu, dále barva, kterou bude hodnota znázorněna a na posledním místě se zde definuje průhlednost hodnoty v grafu (tj. průhlednost samotného bodu znázorňující danou hodnotu).

- **biggestPointWidth -** uživatel má pomocí tohoto parametru možnost definovat maximální povolenou velikost poloměru největšího bodu grafu. Za pomocí tohoto parametru se také přepočítávají na základě jednoduché trojčlenky všechny ostatní poloměry jednotlivých bodů. Jednotkou velikosti tohoto parametru je 1 pixel.

- **where, pictureSize, axisColor, axisWidth, backgrounColor, pad -** tyto parametry již byly všechny popsány (4.4.1 a 4.4.6), jejich význam v tomto modulu je stejný.

Chceme-li vykreslit samotný graf, musíme na instanci třídy PointChartC (jedinný povinný parametr pro vytvoření instance této třídy je parametr values) zavolat metodu **draw()**, která vykreslení zajistí. Metoda draw() pracuje ještě s metodami: **__eraseUsefull()** a **__recalculateValuesForPrint(self, size, values)**. Význam těchto metod byl již popsán v modulu LineChart.py (4.4.6), zde mají stejnou funkčnost.

PointChart.py dále importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1) a z modulu GradientColor importuje třídu **GradientColorC** (4.4.3).

4.4.10 Modul PyramidChart.py

Modul obsahuje třídu **PyramidChartC**, ktrá dědí ze třídy **object**. Tato třída má za úkol generovat požadované pyramidové grafy. Při vytváření instance třídy PyramidChartC máme k dispozici řadu parametrů, jimiž můžeme výsledný graf ovlivnit. Možné parametry jsou: **values, colors, color1, color2, colorEdgeInside, colorEdge, pictureSize, where a backgroundPad**. Všechny parametry již byly popsány ((4.4.1) a (4.4.8)). Třída používá pro pyramidové grafy pozadí trojrozměrných sloupcových grafů, avšak nutně toto pozadí vykreslovat nemusí. Zvolí-li se všechny barvy pozadí jako bílé, modul rozpozná, že pozadí se má ignorovat a začne kreslit pyramidový graf rovnou.

Chceme-li vykreslit samotný graf, je nutno na instanci třídy PyramidChartC (jedinný povinný parametr pro vytvoření instance této třídy je parametr values) zavolat metodu **draw()**. Metoda draw ve svém průběhu volá také všechny ostatní metody třídy PyramidChartC, jsou to: **__drawPyramid(self, im, draw, numberValues)** a **__drawParts(self, numberValues, V1, V2, V3, topOfPyramid, im)**. Význam těchto metod je následující:

- **__drawPyramid(im, draw, numberValues) -** do již nakresleného pozadí (pozadí může být prázdné, nebo typově stejné jako pozadí sloupcových či trojúhelníkových grafů) nakreslí pyramidu s automaticky optimalizovanými, vypočítanými rozměry a tuto pyramidu celou vyplní barvou, která je později předána jako klíčová barva specifikující výměnu pixelů při volání vunkce compositeMask modulu Mask.py. Parametry této metody jsou:
  - im - odkaz na zdrojový obrázek (pro operace s obrázkem jako celkem - tj. uložení, nahrání apod.).
  - draw - také odkaz na zdrojový obrázek (pro operace samotného kreslení).
  - numberValues - zachycuje počet hodnot grafu.
- **__drawParts(numberValues, V1, V2, V3, topOfPyramid, im) -** klíčová metoda modulu Pyramid.py, zde dochází k samotnému výpočtu poměrového zastoupení jednotlivých hodnot v grafu. Zde se z jednotně barevné pyramidy stává barevný graf. Parametry této metody jsou:
  - im - odkaz na zdrojový obrázek (pro operace s obrázkem jako celkem - tj. uložení, nahrání apod.)
  - numberValues - zachycuje počet hodnot grafu.

- topOfPyramid - x-ová a y-ová souřadnice vrcholu pyramidy.
- V1 - Levý dolní vrchol pyramidy.
- V2 - Pravý dolní vrchol (přední stěny) pyramidy.
- V3 - Pravý dolní vrchol (boční stěny) pyramidy.

Všechny vrcholy pyramidy mají pro výsledný graf velký význam. Jejich vlastnostmi a vztahy mezi nimi je ovlivněno samotné vykreslení hodnot grafu.

Modul Pyramid.py dále importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1), **math** (4.4.8), z modulu Background importuje třídu **BackgroundC** (4.4.1), z modulu GradientColor importuje třídu **GradientColorC** (4.4.3), z modulu Mask importuje třídu **MaskC** (4.4.7).

Poznámka: kreslení pyramidového grafu je z hlediska času stráveného generováním grafu nejsložitější ze všech grafů.

4.4.11 Modul RadarBackground.py

Modul obsahuje třídu **RadarBackgroundC**, která dědí ze třídy **object**. Tato třída má na starosti vykreslování pozadí pro radarové grafy. Při vytváření instance této třídy máme k dispozici následující parametry: **maximum, pictureSize, numberGroups, where, backgroundSize, color1, color2, colorEdge a pad**.

Nutno podotknout, že žádný parametr nemá defaultní hodnoty. Pro vytvoření instance této třídy se musí předat pro každý parametr vyhovující data.

Všechny parametry, které třída RadarBackground při vytváření instance používá, byly již popsány, kromě parametrů numberGroups a maximum. Parametr **numberGroups** určuje počet hodnot, které patří k sobě. Máme-li například 3 skupiny, v každé skupině 2 hodnoty, pak bude pozadí pro graf rozděleno na 3 části. Parametrem **maximum** je nejvyšší hodnota v radarovém grafu. Parametr **pictureSize** je v podstatě stejným parametrem jako parametr pictureSize třídy PieChartC (4.4.8). Je tvořen pouze jedním rozměrem kvůli čtvercovému rozměru pozadí, kde stačí zadat jednu stranu - druhá strana je stejná. V parametrech, které ovliňují barvy, chybí (oproti parametrům třídy BackgroundC (4.4.1)) parametr colorEdgeInside. Důvod je ten, že RadarBackgroundC vytváří pozadí pouze dvourozměrné a není potřeba zdůraznit přední stěnu.

Chceme-li vykreslit daný graf, stačí na vytvořené instanci zavolat metodu **draw()**. Tato metoda zajistí vlastní vykreslení pozadí, k vykreslení nepoužívá žádnou jinou metodu. Návratovou hodnotou metody draw() je odkaz na obrázek s požadovaným pozadím grafu.

Modul RadarBackground.py dále importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1) a z modulu GradientColor importuje třídu **GradientColorC** (4.4.3).

4.4.12 Modul RadarChart.py

Modul obsahuje třídu **RadarChartC**, která dědí ze třídy **object**. Tato třída má za úkol vykreslit do již předkresleného pozadí radarový graf. Při vytváření instance třídy **RadarChartC** máme k dispozici následující parametry: **values, gradientColor1, gradientColor2,**

spaceing, pictureSize, where, color1, color2, colorEdge, pad. Všechny parametry již byly popsány (4.4.1,4.4.2) kromě parametru spaceing. Význam je následující.

Parametr **spaceing** se v radarovém grafu projevuje jako vzádlenost mezi jednotlivými hodnotami grafu, přesněji řečeno mezi jednotlivými sloupci grafu. Sloupce však nejsou sloupce v pravém slova smyslu. Jsou to kruhové výseče s různými poloměry. Nutno dodat, že jednotkou vzdálenosti není pixel, jak by se dalo očekávat, ale 1 stupeň.

Chceme-li vykreslit samotný graf, na instanci třídy RadarChartC (jedinný povinný parametr pro vytvoření instance této třídy je parametr values) musíme zavolat metodu **draw()**. Tato metoda k vykreslení grafu využívá ještě metodu **__roundValue(x)** (4.4.1).

Nutno dodat, že barevné přechody obsažené ve sloupcích radarového grafu nejsou vykresleny vždy způsobem startovní barva - konečná barva. Způsob start - cíl je uplatněn pouze u největšího sloupce. Tato vlastnost není závadou. Naopak způsobuje, že na první pohled je rozlišitelné, která hodnota je větší.

Modul RadarChart.py dále importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1), **math** (4.4.8), z modulu GradientFilePie importuje třídu **GradientFilePieC** (4.4.4), z modulu Mask importuje třídu **MaskC** (4.4.7), z modulu RadarBackground importuje třídu **RadarBackgroundC** (4.4.11).

4.4.13 Modul TransparentBarChart3D.py

Modul obsahuje třídu **TransparentBarChart3DC**, která dědí ze třídy **object**. Tato třída má na starosti vykreslovat průhledné sloupcové grafy do již předkresleného pozadí. Možné parametry při vytváření instance této třídy jsou: **values, chartColor, where, pictureSize, backgroundDepth, color1Bg, color2Bg, colorEdgeInsidebg, colorEdgeBg, pad, transparency, barDepth**. Všechny parametry již byly popsány (4.4.2) kromě parametrů chartColor a transparency. Jejich význam je následující:

- **chartColor -** barva sloupců.
- **transparency -** parametr vyjadřující průhlednost jednotlivých sloupců (jak se průhlednosti dosáhne je vysvětleno níže).
- **values, where, pictureSize, backgroundDepth, color1Bg, color2Bg, colorEdgeInsidebg, colorEdgeBg, pad, barDepth** již byly vysvětleny (4.4.2).

Průhlednost je dosažena prostřednictvím modulu **Image**. Ten umožňuje definici nového obrázku prostřednictvím funkce **Image.new(formát, velikost, barva)**. Význam parametrů funkce Image.new():

- **formát -** formát obrázku (např. RGBA)
- **velikost -** velikost obrázku (např (800,600))
- **barva -** jednotná barva obrázku při jeho vytvoření (např. (255, 255, 255) - bílá)

Postup, jak se samotného efektu průhlednosti dosáhne, byl uveden v sekci 3.2.

Chceme-li vykreslit samotný graf, musíme na vytvořené instanci třídy TransparentBarChart3DC (jedinný povinný parametr pro vytvoření instance této třídy je parametr values) zavolat metodu **draw()**. Tato metoda nejprve vykreslí požadované pozadí grafu a poté zavolá metodu **__drawing()**, která se postará o samotné vykreslení grafu do již nakresleného pozadí. TransparentBarChart3DC také obsahuje pomocnou metodu **__roundValue(x)** (4.4.1).

Modul TransparentBarChart3D.py dále importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1), z modulu Background importuje třídu **BackgroundC** (4.4.1), z modulu GradientColor importuje třídu **GradientColorC** (4.4.3).

Poznámka: sloupcové grafy, které modul TransparentBarChart3D vykresluje, jsou od ostatních sloupcových grafů odlišné a to v šířce každého sloupce. Sloupce jsou široké vždy tak, že se navzájem dotýkají. Sloupcové grafy, které vykreslují ostatní moduly se vzájemně nedotýkají.

4.4.14 Modul TransparentGradientBarChart3D.py

Modul obsahuje třídu **TransparentGradientBarChart3DC**, která dědí ze třídy **object**. Tato třída má na starosti vykreslovat parametrizované prostorové průhledné sloupcové grafy s barevnými přechody. Možné parametry při vytváření instance této třídy jsou: **values, gradientColor1, gradientColor2, transparency, where, pictureSize, backgroundDepth, color1Bg, color2Bg, colorEdgeInsidebg, colorEdgeBg, backgroundPad a barDepth**. Všechny parametry již byly popsány (4.4.1, 4.4.2, 4.4.13).

Chceme-li vykreslit samotný graf, musíme na instanci třídy TransparentGradientBarChart3DC (jedinný povinný parametr pro vytvoření instance této třídy je parametr values) zavolat metodu **draw()**. Tato metoda požadovaný graf vykreslí. Draw() ve svém průběhu také využívá metody: **__drawLine(self, (fromXY, toXY), color, width, im)** - nakreslí jednoduchou čáru z bodu A do bodu B specifikované barvy a šířky, **__drawZDimension(values)** (4.4.2), **__makeTransparency(values)**.

Metoda __makeTransparency(values) má za úkol zprůhlednit sloupce grafu dle definovaného parametru transparency třídy TransparentGradientBarChart3DC. Parametr values metody __makeTransparency je seznamem x-ových a y-ových hodnot startovních a koncových bodů slopců grafu. Startovní bod je levý horní vrchol sloupce a koncový bod je pravý spodní vrchol sloupce.

Princip, jak průhlednost funguje, byl uveden v sekci 3.2.

Graf využívá také barevných přechodů, které jsou generovány modulem GradientFileRectangle.py. Princip tvorby dvoubarevného přechodu byl uveden vsekci 3.3.3.

Modul TransparentGradientBarChart3D.py dále importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1), **os**, z modulu Background importuje třídu **BackgroundC** (4.4.1), z modulu BarChart importuje třídu **BarChartC** (4.4.2), z modulu GradientColor importuje třídu **GradientColorC** (4.4.3). Všechny moduly již byly popsány krom modulu **os**. Tento modul je v mém řešení použit, protože umožňuje jednoduchý způsob práce se soubory (např. funkce pro mazání souborů).

4.4.15 Modul TriagnleChart.py

Obsahuje třídu **TriangleChartC**, která dědí ze třídy **object**. Tato třída má na starosti vykreslování parametrizovaných trojúhelníkových grafů. Při vytváření instance této třídy máme k dispozici následující parametry: **values, gradientColor1, gradientColor2, transparency, pictureSize, where, color1, color2, colorEdgeInside, colorEdge a backgroundPad** (4.4.1, 4.4.13).

Chceme-li vykreslit samotný graf, musíme na instanci třídy TriangleChartC (jedinný povinný parametr pro vytvoření instance této třídy je parametr values) zavolat metodu **draw()**. Tato metoda pro správné vykreslení grafu využívá metodu **__roundValue(x)** (4.4.1).

Princip vykreslení trojúhelníkového gradientu s definovanou průhledností je následující:

1. Připravíme si tři stejně veliké obrázky v normálním režimu (např. "**RGBA**") a jeden (také stejně veliký) obrázek s maskou (režim "**L**"), která je od začátku neprůhledná.

2. Do prvního obrázku nakreslíme pozadí.

3. Do obrázku s maskou nakreslíme obdélník s úplnou průhledností a rozměry pozadí prvního obrázku. Tento obdélník nakreslíme na stejné souřadnice, jako má pozadí prvního obrázku.

4. Do druhého obrázku nakreslíme požadované hodnoty grafu, tedy jednotlivé trojúhelníky. Kreslíme barvou, která se pak specifikuje jako barva pro náhradu pixelů v metodě **compositeMask** modulu **Mask.py**. Současně s kreslením trojúhelníků do druhého obrázku kreslíme trojúhelníky se specifikovanou průhledností do obrázku s maskou.

5. Do třetího obrázku si nakreslíme barevné přechody (obdélníkového tvaru) stejné výšky a šířky jako trojúhelníky ve druhém obrázku a jejich podstavy.

6. Zavoláme funkci **compositeMask** modulu **Mask.py**, která šedivou barvu trojúhelníků druhého obrázku nahradí barvou barevného přechodu (třetího obrázku) a výsledek uloží do druhého obrázku.

7. Můžeme smazat třetí obrázek.

8. Do prvního obrázku vložíme druhý obrázek s předlohou vytvořené masky ve čtvrtém obrázku.

9. Výsledek uložíme, ostatní již nepotřebné obrázky (dva, čtyři) můžeme smazat.

Modul TriagnleChart.py dále importuje moduly: **Image** (4.4.1), **ImageDraw** (4.4.1), **os** (4.4.14), z modulu Background.py importuje třídu **BackgroundC** (4.4.1), z modulu GradientFileRectangle.py třídu **GradientFileRectangleC** (4.4.5), z modulu Mask.py importuje třídu **MaskC** (4.4.7).

4.5 Použití API vlastního řešení

Struktura API je velice jednoduchá a jeho použití ještě jednodušší. Popis jak vykreslit graf je následující:

1. Vytvoříme si data, pro která chceme vykreslit graf.
2. Vytvoříme instanci třídy požadovaného grafu.
3. Na instanci třídy požadovaného grafu zavoláme metodu **draw()** a předáme jí data a parametry.

Pro ukázku použití API byl vytvořen speciální testovací modul **Demo.py**. Tento modul obsahuje konkrétní příklady jak vytvořit grafy:

1. Sloupcové s barevným přechodem (4.4.2),
2. spojnicové s orientovanými hodnotami vůči průsečíku os souřadnic (4.4.6),
3. spojnicové s neorientovanými hodnotami vůči průsečíku os souřadnic (4.4.6),
4. koláčové (4.4.8),
5. bodové (4.4.9),
6. pyramidové (4.4.10),
7. radarové (4.4.12),
8. průhledné jednobarevné sloupcové bez odskoku sloupců (4.4.13),
9. průhledné sloupcové s gradienty, s odskoky sloupců (4.4.14),
10. průhledné trojúhelníkové (jen 2D) s gradienty a odskoky mezi sloupci (4.4.15).

Ukázka kódu modulu **Demo.py** je zobrazena ve výpise zdrojového kódu 1.

```
def pyramidChart():
  values=(10,20,30,40,50,60)
  chart = PyramidChartC( values )
  chart.draw()

def radarChart():
  values=(( 50, 100, 150),( 77, 70, 90 ),( 33, 78, 99 ))
  color1 = "purple"
  color2 = " lightblue "
  chart = RadarChartC( values, color1, color2 )
  chart.draw()
```

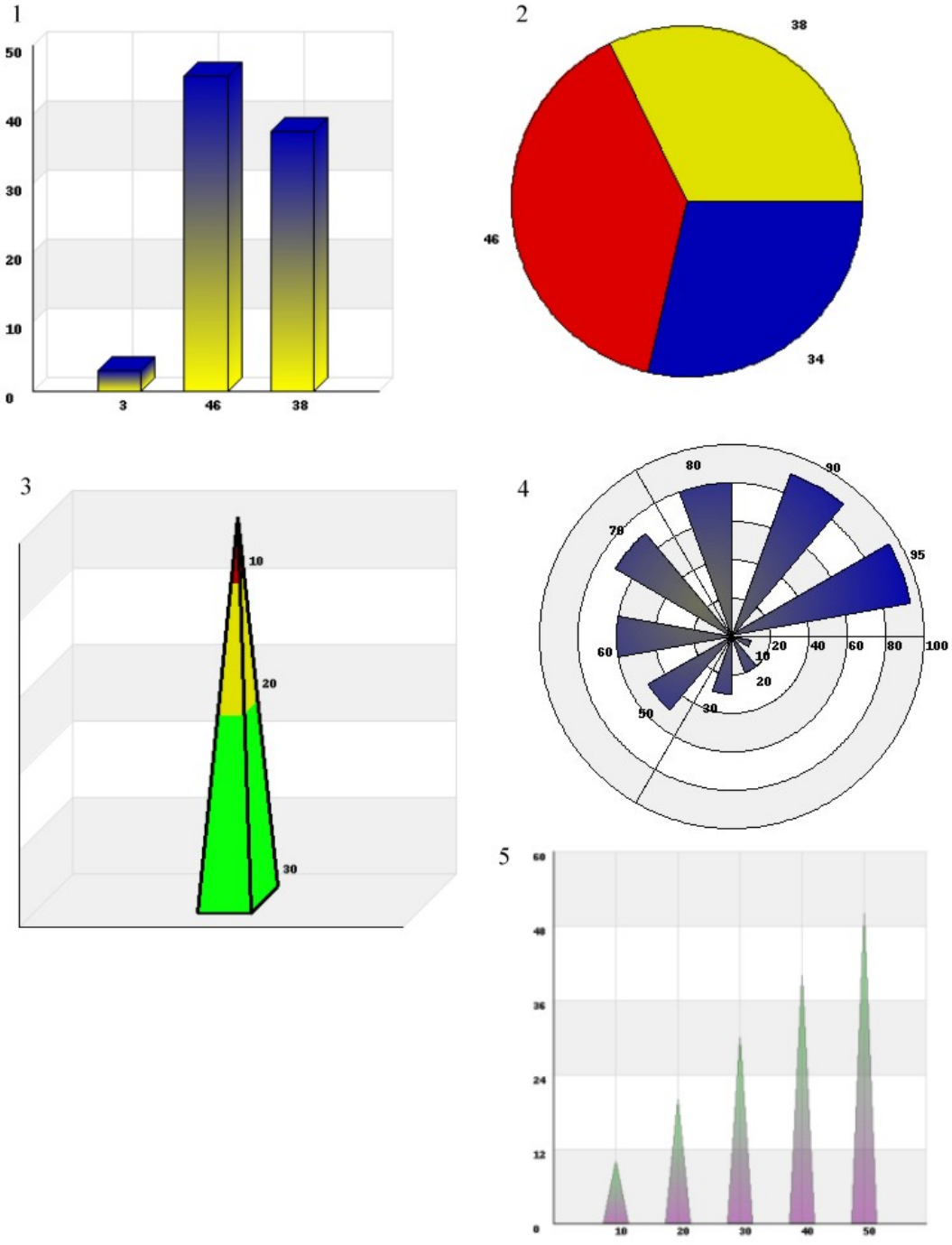
Výpis 1: Ukázka dokumentace metody

Modul Demo.py importuje všechny třídy ze všech modulů vlastního řešení. Dále importuje moduly: **os** (4.4.14) a **warnings**. Jediný dosud nepopsaný modul je modul warnings. Tento modul je importován z důvodu skrytí varovných hlášení, která vznikají drobnými nepřestnostmi ve výpočtech týkajících se samotného kreslení grafů. Jiný důvod zde nemá.

Poznámka: po spuštění modulu Demo.py se vytvoří ke každému modulu stejnojmenný soubor s koncovkou **pyc**. Tento soubor obsahuje přeložený obsah prvotního stejnojmenného modulu. Při opětovném spuštění modulu se již využívá tento přeložený soubor namísto prvotního. Důvodem tohoto faktu je rychlejší průběh. Nepřeložený modul se musí přeložit a pak spustit, přeložený soubor se jen spustí.

4.6 Ukázky grafů vlastního řešení

Na obrázku 9 jsem uvedl některé ukázky grafů, které nově vytvořená knihovna kreslících funkcí generuje.



Obrázek 9: Ukázky grafů, které generuje vlastní knihovna.

5 Dokumentace

Pro doukumetaci celé knihovny byl zvolen nástroj **epydoc**. Tento nástroj je **freeware**, volně dostupný na internetu [3]. Epydoc generuje vzhledově příjemnou dokumentaci (design podobný javovské dokumentaci generované nástrojem javadoc) nejen ve **formátu html** dokumentů, ale i v jiných formátech (vlastní řešení je zdokumentováno ve formátu html stránek).

Pro správné vygenerování je třeba dodržovat určitá **pravidla**:

1. Za definicemi metod a tříd bychom měli uvádět dokumentační řetězce (příklad dokumentačního řetězce: """Toto je dokumentační řetězec""")
2. U metod popisovat typ jejich parametrů (např.: @type parametr1: number)
3. U metod popsat, co který parametr znamená (např.: @param parametr1: Definuje odskok)
4. Popisovat typ návratové hodnoty (např.: @rtype: number)
5. Popisovat, co návratová hodnota znamená (např.: @return: Sum of all values)

Příklad jak správně zdokumentovat metodu **__roundValue(x)** nalezneme ve výpise zdrojového kódu (2).

```
def __roundValue( self , x ):
    '''
    Method which round specified value.

    @type self: reference
    @param self: Reference to itself.
    @type x: number
    @param x: Number to round up.
    @rtype: number
    @return: Rounded value of specified number.
    '''
    .
    .
    .
    #source code
    .
    .
```

Výpis 2: Ukázka dokumentace metody

6 Demonstrační web

O požadavky uživatelů se stará samostatný modul - **webserver.py**. Přicházející požadavky ve formě **URL** (Unified resource language) převede tento modul na dvě části:

1. Cestu k souboru.
2. Parametry (nutné i doplňující) ovlivňující graf.

Cesta k souboru slouží k otevření požadovaného dokumentu umístěného na serveru (v našem případě ./index.html).

Soubor **index.html** je jediným dokumentem, který umí serverový modul webserver.py vracet žadatelům jako odpověď (krom chybových hlášení). Obsahuje základní strukturu html dokumentu (tagy html, body, head apod.), jednoduchý text a odkaz na soubor pojmenovaný **obrazek.jpg** (do tohoto obrázku se vykreslují žadateli definované obrázky).

Princip fungování webserveru je následující:

1. Žadatel pošle přes svůj internetový prohlížeč dotaz na dokument (index.html) umístěný na serveru s požadovanými hodnotami a parametry pro graf.
2. Modul webserver.py dotaz zachytí, zpracuje a pokud jsou parametry syntakticky správně, vygeneruje graf do souboru obrazek.png.
3. Pak náš modul pošle na adresu žadatele odpověď v podobě html dokumentu (index.html), obsahujícího krom čistého textu požadovaný obrázek s grafem (pokud je URL adresa zadána chybně, je zaslána chybová hláška).

Odpovědi serveru mohou být:

1. V případě, kdy je napsána URL adresa bez syntaktických chyb a data jsou ve správném tvaru, pak se jako odpověď posílá soubor index.html, který obsahuje požadovaný obrázek.
2. V případě, kdy je napsána URL adresa bez syntaktických chyb a data jednotlivých parametrů nejsou ve správném tvaru, posílá se chybové hlášení, které říká, že jedna z hodnot parametrů není ve správném tvaru.
3. V případě, že je v URL adrese chyba (nejčastější chybou je špatné pořadí parametrů) posílá se jako odpověď chybová hláška, která chybu specifikuje.

Jediný způsob jak získat odpovědí požadovaný graf, je zadat správnou URL adresu (samozřejmě s parametry) bez chyb.

7 Závěr

Byla prozkoumána, zhodnocena a částečně popsána dostupná stávající řešení problému vykreslování grafů. Byla vytvořena také vlastní knihovna řešící problematiku vykreslování kvalitně vypadajících grafů. Grafy, které knihovna generuje, jsou vysoce variabilní, kvalitně vykreslené, nechybí jim možnost definovat průhlednost, barevné přechody, velikosti celkového obrázku, odskoky od okrajů, barvy pozadí a mnoho dalších. Všechny typy grafů (v reálných velikostech - rozlišení kolem velikosti 1200x800 pixelů) jsou generovány v čase kolem 1 vteřiny.

Princip struktury a funkcionality zdrojových souborů vlastního řešení byl zdokumentován. Pro ukázku nově vzniklé knihovny byl také vytvořen demonstrační web.

Bakalářská práce byla zadána nezávazně na jiných projektech. Další vývoj spojený s použitím či rozšířením již stávající knihovny je možný. Použití modulů knihovny je velice jednoduché, další rozšíření je možné, zejména o nové typy grafů, popř. vytvoření doplňků k stávajícím grafům (jako např.: vylepšení popisků, vytváření legendy apod.).

Radek Daníšek

8 Reference

[1] Daryl Harms, Kenneth McDonald, *Začínáme programovat v jazyce Python - 2.opravené vydání,* Brno: Computer Press, a.s. , 2008, ISBN 978-80-251-2161-0.

[2] PyChart, *http://home.gna.org/pychart*, ke dni 30.4. 2008.

[3] Epydoc, *http://epydoc.sourceforge.net*, ke dni 30.4. 2008.

[4] Python Imageing Library, *www.pythonware.com/products/pil*, ke dni 30.4. 2008.

[5] Wikipedia, the free encyclopedia, *http://en.wikipedia.org/wiki/Alpha_compositing*, ke dni 30.4. 2008.